(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
21 November 2002 (21.11.2002)

PCT

(10) International Publication Number
WO 02/093343 A2

(51) International Patent Classification[7]: G06F 3/00

(21) International Application Number: PCT/EP02/04837

(22) International Filing Date: 3 May 2002 (03.05.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
01111834.6 16 May 2001 (16.05.2001) EP

(71) Applicant *(for all designated States except US)*: **INTERNATIONAL BUSINESS MACHINES CORPORATION** [US/US]; New Orchard Road, Armonk, NY 10504 (US).

(71) Applicant *(for LU only)*: **IBM DEUTSCHLAND GMBH** [DE/DE]; Pascalstrasse 100, 70569 Stuttgart (DE).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **BAITINGER, Friedemann** [DE/DE]; Feldbergstr. 15, 71131 Jettingen (DE). **KREISSIG, Gerald** [DE/DE]; Schuhgasse 2, 71083 Herrenberg (DE). **SAALMÜLLER, Jürgen** [DE/DE]; Keltenburgstr. 36, 71034 Böblingen (DE). **SCHOLZ, Frank** [DE/DE]; Kapellenbergstr. 95, 71120 Grafenau (DE).

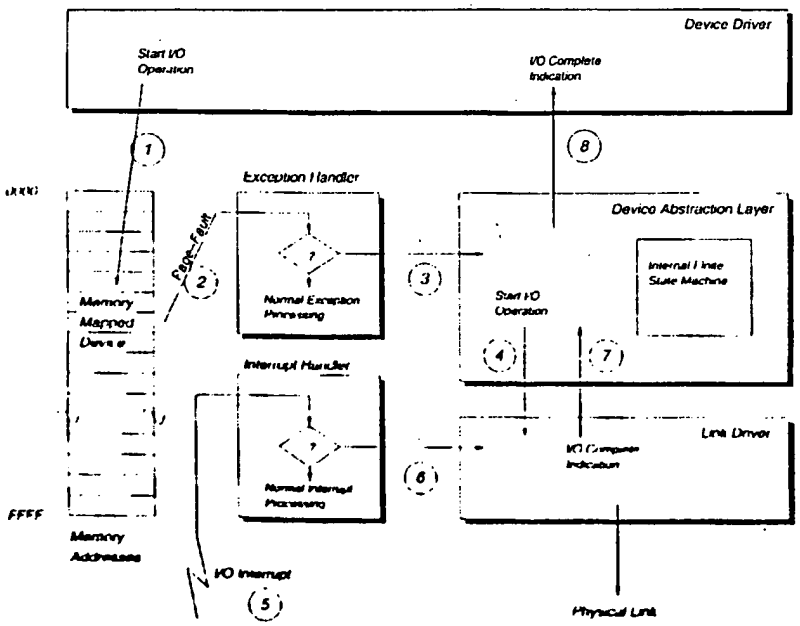
(74) Agent: **TEUFEL, Fritz**; IBM Deutschland GmbH, Intellectual Property, Pascalstr. 100, 70548 Stuttgart (DE).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent

[Continued on next page]

(54) Title: METHOD AND SYSTEM FOR EFFICIENT ACCESS TO REMOTE I/O FUNCTIONS IN EMBEDDED CONTROL ENVIRONMENTS



(57) Abstract: A method for accessing I/O devices in embedded control environments is provided, wherein said I/O devices are remotely attached to an embedded microprocessor. By mapping said I/O devices resources to said microprocessor s address or memory address space, existing device drivers can be reused and the time-to-market capability is greatly improved.

WO 02/093343 A2

(BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

D E S C R I P T I O N

Method and System for Efficient Access to Remote I/O Functions in Embedded Control Environments

Field of the Invention

The present invention relates, in general, to accessing I/O functions. More specifically, the invention relates to accessing I/O functions that are remotely attached to a microcontroller. Still more specifically, the present invention deals with accessing such remotely attached I/O functions in embedded control environments.

Background of the Invention

In distributed embedded control environments I/O engines are often remotely attached to the embedded controller through a serial link. Typically access to these devices is facilitated by directly programming the serial link controller hardware.

Fig. 1 shows such an environment where an embedded controller needs to operate a series of I/O devices which are attached at the remote end of a link.

At the lowest level a microprocessor accesses I/O resources either through its I/O address space for architectures such as the Intel x86, or through its memory address space for architectures such as the IBM PowerPC. The software entity that controls the hardware at this level is usually called a Device Driver. Towards higher level software applications the Device Driver provides a generic view of the device. In UNIX systems, it is typically visible in the hierarchical file-system name space in the /dev tree and access to it is

facilitated using regular file-system semantics and interfaces like:

- open();
- close();
- read();
- write();
- ioctl()

In embedded control applications it is not always possible to attach all I/O devices such that they can be directly addressed at the microprocessor bus level as shown in Fig. 2. The I/O functions are often implemented in a standalone ASIC which can be attached to the embedded controller's microprocessor via some link hardware. Consequently, the resources of the I/O devices are not directly visible in the I/O address space or memory address space of the microprocessor. What is visible to the microprocessor though are the resources of the link hardware that connects the I/O ASIC with the embedded controller.

Since no existing device driver can be used in this situation the full software path to the I/O devices needs to be facilitated through custom device interface software. The need for this custom device interface software is a limiting factor for time-to-market which is often key in the embedded control market nowadays.

In addition custom implementation of the device driver layer increases the probability of shipping software bugs to the field which cannot be easily recovered from. Unlike regular PC applications where it is relatively simple to download a new version of a previously buggy application over the Internet, it is not easily possible to replace faulty software in embedded control applications.

Until recently this problem did not exist because embedded control applications traditionally used custom operating systems, custom device drivers, and custom hardware. Since the problem is rather new, standard solutions are not yet available, however, with the emerging push towards standardization in embedded control environments new solutions will become more prevalent.

A major driver towards this standardization is Linux and the overall OpenSource movement.

Summary of the Invention

It is therefore an object of the present invention to provide a method for accessing I/O devices in embedded control environments thereby greatly reducing the programming effort.

This and other objects and advantages will be achieved by the method disclosed in claim 1.

Advantageous embodiments are laid down in the dependent claims.

Brief Description of the Drawings

The invention will in the following be described in more detail in conjunction with the drawings, in which

Fig. 1 schematically shows an I/O card remotely attached to a microprocessor according to the state of the art;

Fig. 2 schematically depicts a microprocessor bus attached I/O;

Fig. 3 schematically shows a hardware solution according to the invention;

Fig. 4 schematically depicts a software structure with a Device Abstraction Layer (DAL) according to the present invention; and

Fig. 5 schematically depicts a sequence of events describing the DAL operation according to the invention.

Detailed Description of the Preferred Embodiment

The present invention allows for reuse of existing device drivers which are available for common I/O devices as long as they are mapped to the processor's I/O- or memory-address space in well known locations and with a well known layout. Reusing existing device driver code greatly improves the time-to-market capability and it reduces the software test effort as well.

It has to be mentioned that the present invention does not change the existing device drivers but adapts them to a fictitious device.

For remote but well known device types such as serial ports which are often referred to as UARTs (Universal Asynchronous Receiver/Transmitter) where device drivers exist for nearly all operating systems and Real Time Control Programs (RTOS), a Device Abstraction Layer (in the following called „DAL") is provided which maps the devices' resources into the microprocessor I/O address space or memory address space as if the remote device was locally available to the microprocessor.

Table 1 shows an example of UART resources as they are visible to the microprocessor as registers either in the I/O address space or the memory address space. By doing so, it is possible to instantly reuse existing device drivers and all the existing application software that uses them without the need to rewrite a vast amount of complex code.

Table 1 UART Registers

Offset	R/W	Description
0	R	Receive Buffer
	W	Transmitter Holding Register
1	R/W	Interrupt Enable Register
2	R	Interrupt Identification Register
3	R/W	Line Control Register
4	R/W	Modem Control Register
5	R	Line Status Register
6	R	Modem Status Register
7		Unused
8	R/W	Divisor Latch (LSB)
9	R/W	Divisor Latch (MSB)

The implementation of the DAL is done in either one of the following ways:

1. Embedding control hardware to redirect the requests and responses over the link to the remote device and still maintain the device abstraction.

2. Using the microprocessor's memory management unit to cause a program exception as soon as the DAL's resources are accessed which is done by providing a thin layer

device abstraction which executes in the context of the exception handler.

As to the hardware solution, figure 3 shows how a microcontroller 2 or equivalent processor core within an ASIC device accesses the link logic 6 by a system bus 4 via a local register set 8 combined with a state machine. The link logic 6 may communicate with a respective link logic 10 of the remote device, e.g., by using an appropriate and suitable protocol via link 16. The link logic 10 connects to a remote register set 12 containing a second state machine and attaches to the remote device 14 via a bus 18. It has to be mentioned that the link logic 6 can be any kind of a data exchange (data link) between two stations A and B. This may be achieved by a simple serial interface, a LAN, a private network or a global network, etc., but is not restricted thereto.

The microcontroller 2 triggers, e.g., a read or write operation to the remote device 14 by writing to the local register set 8. This in turn activates the state machine in 8 and controls the transfer of data to the link logic 6. The link logic units 6 and 10 transfer the request posted in 8 to the register set 12 and subsequently activate the respective state machine to execute the requested operation in 14, i.e., to write or read any data to/from the remote device 14.

After completion of the operation the state machine 12 triggers a response to be sent back via the link 16 and stored in a register of unit 8. This action may also cause a notification to the micro controller, e.g., an interrupt. Units 2, 6 and 8 may be designed as one single ASIC device integrating all functions or by use of standard vendor components that are interconnected as discrete modules.

The same approach is valid for the units 10, 12 and 14. While the link hardware 6, 10, 12 and 16 is required to overcome the physical distance between units 2 and 14, unit 8 may translate partially or fully to a software layer below the DAL. This allows for cost tradeoffs between hardware and software.

Solution 1 mentioned above is used for rather simple I/O controllers where the hardware design effort is affordable and where performance requirements dictate it.

Solution 2 is a software solution and is used where the hardware cost cannot be afforded but one can live with the overhead of entering and leaving the exception handler context. This solution will be described in more detail in the following.

Fig. 4 schematically depicts a software structure with a Device Abstraction Layer (DAL) according to the present invention.

The general mechanism used by the DAL implementation is a means for generating an exception during instruction execution upon accessing a virtual resource that does not physically exist.

Two methods may be used to trigger the DAL, i.e.,

1) Use of the microprocessors's management unit's capability to generate an exception upon accessing a memory location where no real memory is mapped to; and

2) Use of the microprocessor's execution unit's capability to generate an exception upon execution of a privileged instruction.

The following description of the DAL operation is based on method 1 but is also applicable to methods 2 and 3.

The Device Abstraction Layer "DAL" facilitates the virtualization of the remotely attached device and makes it appear to device driver software as if it was locally attached to the microprocessor bus.

The following sequence of events as shown in Fig. 5 describes the operation of the DAL according to the described software solution and shows how it interacts with the rest of the system:

During steady state operation, at some point in time the device driver for the virtual device may want to issue an I/O operation to the remote device. The device driver writes to the appropriate memory area in the microprocessor's memory address range (Step 1).

The Memory Management Unit of the microprocessor has been programmed such that read/write accesses to the relevant address range cause a Page-Fault Exception (Step 2).

Subsequently, the Page-Fault Exception Handler is invoked and a decision is made whether the exception needs to be handled by the DAL or treated like a regular page-fault.

The Exception Handler transfers control to the DAL (Step 3). It provides sufficient information for the DAL to decode the operation that the device driver actually wanted to perform with the device.

The DAL updates its internal state machine accordingly and issues the Start-I/O operations (Step 4) to the real device over the Link Driver as needed.

Once the I/O operation is complete at the remote device, an I/O complete interrupt is signalled to the microprocessor (Step 5).

Now, the interrupt handler decodes the interrupt source and passes control to the I/O complete handler in the Link Driver (Step 6).

The I/O complete handler decodes the event and passes DAL related events back to the DAL (Step 7).

The DAL retrieves the I/O information from the device via the Link Driver, updates its internal finite state machine tables and signals an I/O complete event to the device driver (Step 8).

The device driver issues read/write operations to the memory map of the device in order to obtain status of the I/O operation. Access to these memory locations again causes page-fault exceptions which are fed to the DAL which knows the relevant information from the I/O device already and thus can make it available to the device driver by completing the interrupted memory read/write operations.

C L A I M S

1. Method for accessing I/O devices in embedded control environments, said I/O devices being remotely attached to an embedded microprocessor,

   characterized by the step of

   mapping said I/O devices' resources to said microprocessor's address or memory address space.

2. Method according to claim 1, characterized in that said I/O devices are selected from the group consisting of Universal Asynchronous Receiver/Transmitter (UART), Universal Serial Bus (USB), Joint Test Action Group (JTAG), and IC Bus (I^2C).

3. Method according to claim 1 or 2, characterized in said mapping step is performed by a device abstraction layer (DAL).

4. Method according to claim 3, characterized in that said DAL is implemented via extra embedded control hardware.

5. Method according to claim 3, characterized in that said DAL is implemented via software.

6. Method according to claim 4, characterized in that said extra embedded control hardware redirects requests and responses over a link to the remote device.

7. Method according to claim 5, characterized in that said DAL uses said microprocessor's memory management unit to cause a program exception by an exception handler.

8. Method according to claim 7, characterized in that a thin layer device is provided which executes in the context of said exception handler.

9. Method according to claim 7 or 8, characterized in that an exception is generated during instruction execution upon accessing a virtual resource unit.

10. Method according to claim 9, characterized in that the microprocessor's management unit is used to generate an exception upon execution of a privileged instruction.

11. A computer system for accessing I/O devices in embedded control environments, said I/O devices being remotely attached to an embedded microprocessor, characterized in that said system comprises means for mapping said I/O devices' resources to said microprocessor's address or memory address space.

12. Computer system according to claim 11, characterized in that said means is a device abstraction layer (DAL).

13. Computer system according to claim 12, characterized in that said DAL is implemented via extra embedded control hardware.

14. Computer system according to claim 12, characterized in that said DAL is implemented via software.

15. Computer system according to claim 14, characterized in that said software is adapted to cause a program exception as soon as the DAL's resources are accessed.

16. Computer system according to claim 15, characterized in that a thin layer device abstraction is provided adjusted to cause said program exception.

17. A computer program product stored on a computer usable medium, comprising computer readable program means for causing a computer to perform a method according to any one of claims 1 to 10.

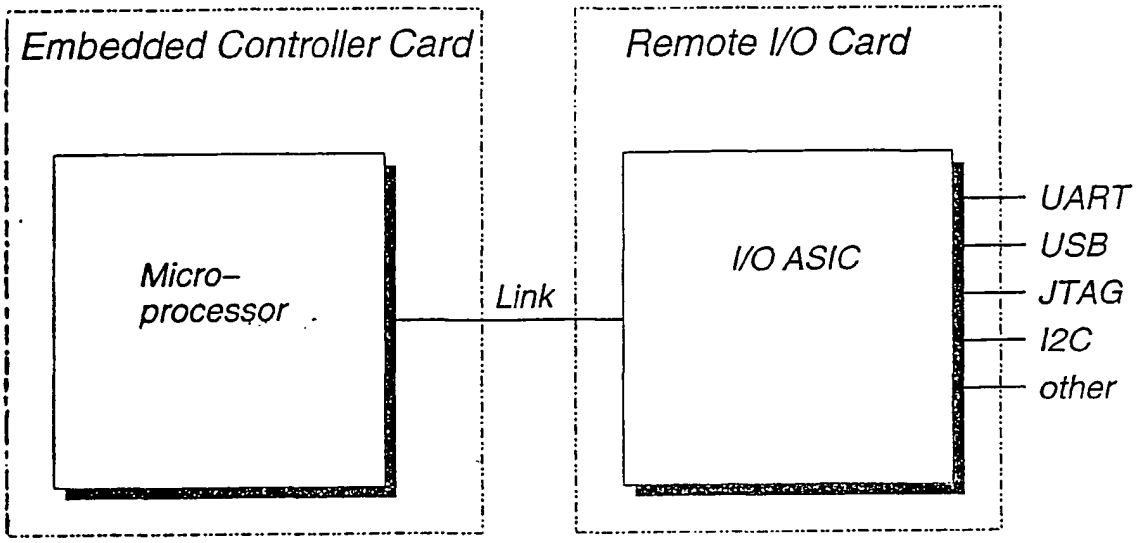

Embedded Controller Card
Remote I/O Card
Micro–
processor
Link
I/O ASIC
UART
USB
JTAG
I2C
other

FIG. 1

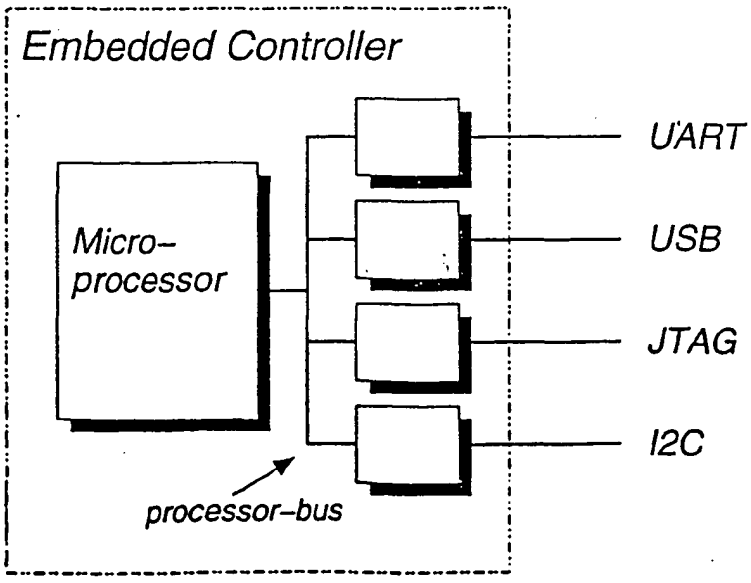

Embedded Controller
Micro–
processor
UART
USB
JTAG
I2C
processor–bus

FIG. 2

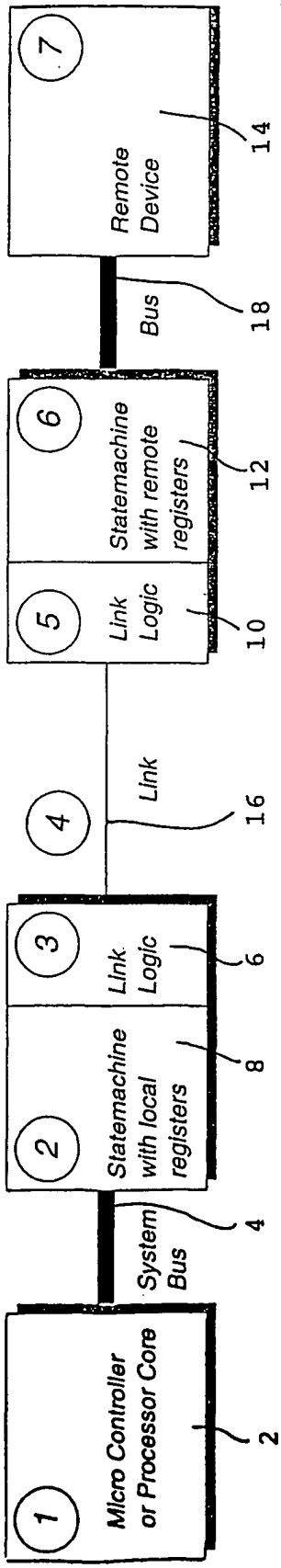

1
Micro Controller or Processor Core
2
System Bus
4
2
Statemachine with local registers
8
3
Link Logic
6
4
Link
16
5
Link Logic
10
6
Statemachine with remote registers
12
Bus
18
7
Remote Device
14

FIG. 3

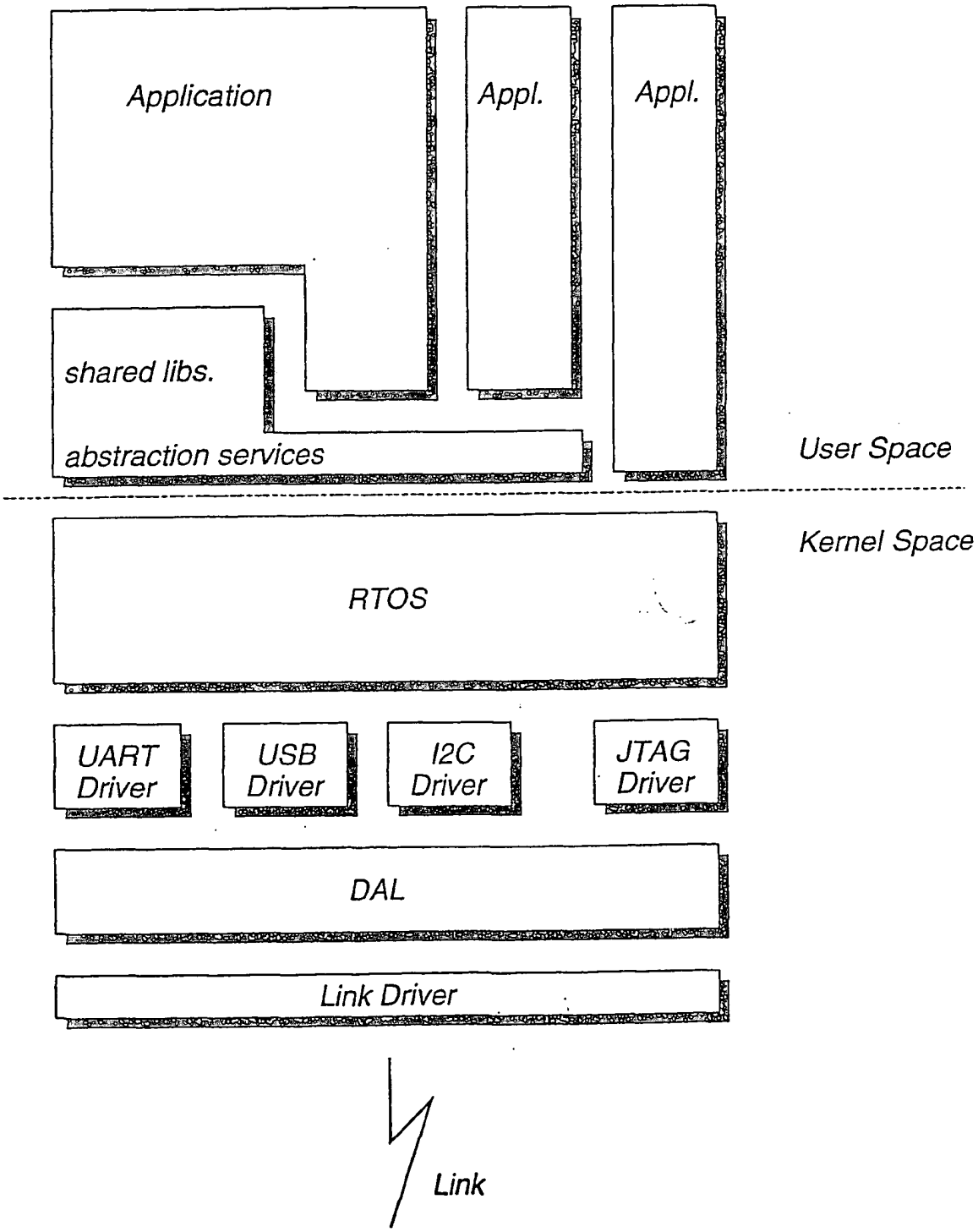

Application
Appl.
Appl.
shared libs.
abstraction services
User Space
Kernel Space
RTOS
UART Driver
USB Driver
I2C Driver
JTAG Driver
DAL
Link Driver
Link

FIG. 4

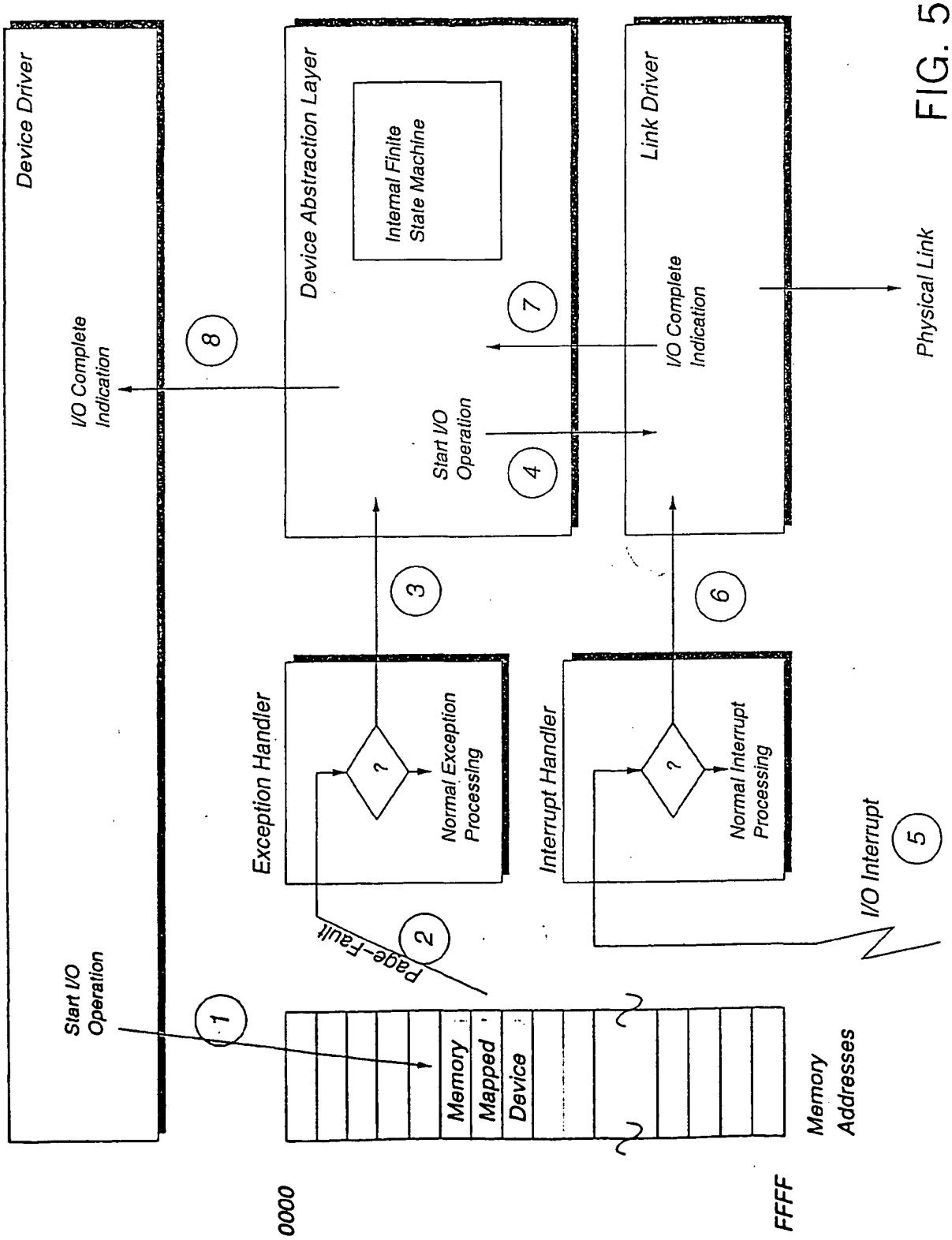

Device Driver
I/O Complete Indication
8
Device Abstraction Layer
Internal Finite State Machine
7
Start I/O Operation
4
Link Driver
I/O Complete Indication
Physical Link
3
6
Exception Handler
?
Normal Exception Processing
Interrupt Handler
?
Normal Interrupt Processing
I/O Interrupt
5
Page-Fault
2
Start I/O Operation
1
Memory Mapped Device
0000
FFFF
Memory Addresses

FIG. 5